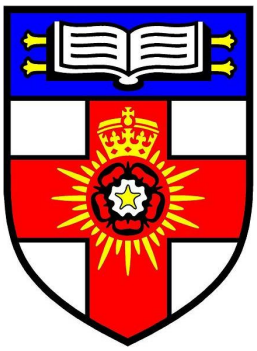
Computer Security: A Machine Learning Approach

Sandeep V. Sabnani

Technical Report
RHUL-MA-2008-09
07 January 2008

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Computer Security: A Machine Learning Approach



Sandeep V. Sabnani

Supervisor: Professor Andreas Fuchsberger

Submitted as part of the requirements for the award of the MSc in Information Security at Royal Holloway, University of London

I declare that this assignment is all my own work and that I have acknowledged all quotations from the published or unpublished works of other people. I declare that I have also read the statements on plagiarism in Section 1 of the Regulations Governing Examination and Assessment Offences and in accordance with it I submit this project report as my own work.

I would like to dedicate this thesis to my loving parents and my late Mama, Mr. Ranjeet Bhagwanani...

Acknowledgements

I wish to acknowledge the support and help of several people who have been instrumental in some way or the other for making this dissertation.

First and foremost, I would like to thank my parents for their never-ending support and encouragement throughout my education. My father has been an invaluable source of inspiration for me always. My mother has always been my biggest motivation in times of high and low. A special thanks to my Mami, Mrs. Gunjan Bhagwanani for her encouragement and for always being there when I need her.

I would like to thank my supervisor Professor Andreas Fuchsberger for his patience, guidance and constant encouragement. It has been a great privilege to work under him. I also wish to thank Mr. William Rothwell for his help and guidance.

I also wish to acknowledge my closest friends Manish Sachdev, Bharat Raisingani, Mohit Vazirani, Sanjay Rohra, Tarun Rochiramani, Lokesh Jagasia, Bimal Sadhwani, Monesh Punjabi and Arvind Sabharwal with whom I have spent the most amazing years of my life and have learned a lot, both academically and personally.

Finally, I would like to extend a very special thanks to my sister Shefali and my brother Pankaj with whom I have shared a very special bond and who have always been instrumental in motivating me in all possible ways.

Abstract

On *Computer Security: A Machine Learning Approach* (Under the supervision of Professor Andreas Fucshberger)

In this thesis, we present the application of machine learning to computer security, particularly to intrusion detection. We analyse two learning algorithms (NBTree and VFI) for the task of detecting intrusions and compare their relative performances. We then comment on the suitability of the NBTree algorithm for the intrusion detection task based on its high accuracy and high recall. We finally state the usefulness of machine learning to the field of computer security and also comment on the security of machine learning itself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation presents concepts from the field of Artificial Intelligence and Machine Learning which can be used to address some of the challenging problems faced in the computer security domain.

Securing computer systems has become a daunting task these days with the rapid growth of the internet (both in terms of size and scalability) and the increasing complexity of communication protocols. The raging war between the security perpetrators and information security professionals has become more intense than ever. New and complicated attack methods are being developed by attackers at an alarming rate by taking advantage of the intricate behaviour of today's networks. CERT has reported 8064 new vulnerabilities in the year 2006 and this figure has been significantly increasing over the past few years [CER07].

Although proactive means for achieving security[1] have existed for a long time, these approaches target the prevention and/or detection of only *known* attacks. *Novel* attacks have been posing a long-standing problem in the field of information security and as such, have received considerable attention in the recent past. Many such novel attacks are difficult to detect solely on the basis of analysis of basic behaviour of communication protocols (in the case of a network) or analysis of basic system calls (in the case of a single host). For instance, an attack might be developed which operates in stealth mode, i.e. it may hide its presence and evade detection [WS02]. Also, increasing complexity of cryptographic mechanisms (like IPSec) has made this detection problem more severe.

Another important problem in the field of computer security that has been present for quite some time is that of *insider* threats. An employee at an organi-

[1] By security, we refer to both security of the network and security of a particular host in the network.

sation is considered to be a *trusted* user and the possibility of an attack from an insider is considered less probable. However, a study by CERT [CER05] showed that insider attacks have been a cause of a lot of tangible and intangible losses to many institutions in the recent past. Many of the insider threats may be unintentional, nevertheless it has become essential to ensure that insider behaviour is in sync with the security policy of the organisation.

Handling the above issues is quite expensive for organisations. The detection task requires security experts to formulate efficient rules for anomaly detection. They are also required to handle large amounts of data. This adds an element of unreliability and also makes the entire process quite slow. Also, checking whether compliance to security policy by insiders is being achieved, is an onerous task for an administrator as normal behaviour changes over time.

Machine Learning has been one of the most promising advancements in Computer Science in the past few decades and has found success in solving intricate data classification problems. It primarily deals with constructing computer programs that automatically improve with experience [Mit97]. The *learning experience* is provided in the form of data and actual *learning* is achieved with the help of algorithms. The two main tasks that are addressed by machine learning are the ability to *learn* more about the given data and to *make predictions* about new data based on learning outcomes from the learning experience [Mal06]; both of which are difficult and time-consuming for human analysts. Machine learning is thus, well-suited to problems that depend on rare, expensive and unreliable *human* experts. It has been successfully applied to complex problems ranging from medical diagnosis to stellar analysis.

The task of detecting intrusions can be considered as a machine learning task as it involves the classification of behaviour into user and adversary behaviour. In this thesis, we study some significant machine learning approaches towards solving some challenging computer security issues (mainly relating to detecting intrusions) which are described in later chapters.

The motivations behind choosing this dissertation topic are manifold. Firstly, the MSc course was instrumental in helping me have a decent understanding of security concepts. The network security lectures gave an overview of anomaly detection and this fuelled my desire to learn more about it. Being a computer engineering graduate and having studied machine learning, I felt this topic would help me apply my knowledge of security to learning and vice-versa. I also felt that it would complement my future plans of pursuing research relating to these two areas.

The following is roughly the structure of the thesis: In the next chapter, we look at some computer security fundamentals and describe the aspects of computer security that are of importance for this thesis. The following chapter introduces some basics of machine learning. Thereafter, we look at how machine learning algorithms can be efficiently used to solve the intrusion detection task by using Weka, a machine learning tool described in [WF05]. This task would demonstrate the use of two algorithms to solve the intrusion detection problem and analyse their relative performance. We then discuss the suitability of one of the algorithm over the other based on their performance metrics. Finally, we conclude with a short discussion on the suitability of machine learning to security, pointers for future work and security of machine learning itself.

Chapter 2

Computer Security

This chapter describes basic security concepts including the requirements of computer security and the ways in which it can be achieved. It also describes the basic types of intrusion detection mechanisms and their current state in achieving the goals of computer security.

2.1 Computer Security Fundamentals

A computing environment comprises of various hardware and software components (also referred to *assets*[1]). Computer security(S) involves the protection of such assets. It can be expressed as a function of confidentiality(C), integrity(I) and availability(A) of information to authorised users.

Ideally, security professionals attempt to maximize the value of S depending on the security requirement of an organisation or application. The following sections describe these concepts in a more detailed manner as given in [PP03] and [Gol99].

2.1.1 Confidentiality

Confidentiality deals with the *secrecy* or *privacy* of assets. It ensures that only authorised users are allowed to access computer assets. This 'access' incorporates any kind of access including reading, writing, printing or even the knowledge that a particular asset exists. In short, as quoted from [PP03], confidentiality means that "only authorised people or systems can access protected data".

[1]Here, we do not talk about assets like human and other intangible assets.

2.1.2 Integrity

The concept of integrity makes sure that assets can only be modified by authorised users. Modification of an asset may include tasks like changing, deleting, writing, changing status and creating. According to Clark and Wilson [CW87], integrity is maintained when

> "No user of the system, even if authorised, may be permitted to modify data items in such a way that assets or accounting records of the company are corrupted."

According to the Orange Book [NIS85], integrity may also be defined as ensuring external consistency.

2.1.3 Availability

This property is concerned with the proper availability of information or services to authorised users whenever desired. It primarily aims to prevent any *denial of service* [Gol99].

Apart from the above properties, there are other properties which may be considered a part of computer security. These include authentication, accountability, reliability, fault-tolerance and assurance [Cra06].

2.2 Challenges in Computer Security

Ideally, a computer system can be made perfectly secure if all the above mentioned properties are *well*[1] satisfied. However, in practice it is impossible to design a system with perfect security and usability [Mal06]. Any system can be subjected to breaches of confidentiality, integrity and/or availability thereby rendering itself in an insecure state. In order to address this scenario, it is acknowledged that a system might fail and so there is a need to put in detection and response mechanisms in addition to the protection mechanisms [Mal06].

2.2.1 Protection

The proactive part of security consists of protecting the asset. The asset is protected in order to prevent any breaches of confidentiality, integrity or availability.

[1] This may be relative to a particular security requirement.

2.2.2 Detection

Since perfect security cannot be achieved, we anticipate that the protection measures might not be able to protect the assets under all cases. This leads to the adoption of detection measures in security. These measures are used detect possible breaches of security and their efficacy depends on the time taken to detect. This time may be different for different assets and may be proportional to the value of the asset. Another factor that contributes to the efficiency of a detection mechanism is the number of false alarms it generates. A false alarm may be a *false positive* or a *false negative*. The higher the number of false alarms, the slower is the detection process and is more expensive.

2.2.3 Response

Supplementing the detection process is the process of responding to security breaches. The response type may be different in different scenarios and would depend on the exact security requirement. Typical response types include evaluating the damage, recovering from the damage, improving with experience, etc.

2.3 Intrusion Detection

Intrusion Detection is used to detect violation of a security policy of an organisation. These violations may be caused by people external to the organisation (i.e. *attackers*) or by employees of the organisation (i.e. *insiders*). Although progress has been made to detect violations by attackers, insider violations are difficult to detect.

2.3.1 Motivations behind Intrusion Detection

Intrusion Detection has received considerable motivation owing to the following reasons as quoted from [Sta06]:

1. If an intrusion is detected quickly enough, an intruder can be identified quickly and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less is the amount of damage done and more quickly that recovery can be achieved.

2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusion.

3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

2.3.2 Goals of Intrusion Detection

Along with the motivations, the goals of intrusion detection can be summarised as below:

1. Detect as many type of attacks as possible including those by attackers and those by insiders.

2. Detect as accurately as possible thereby minimising the number of false alarms.

3. Detect the attacks in the shortest possible time.

2.3.3 Types of Intrusion Detection

In the past few decades, the above requirements have triggered the development of different types of intrusion detection techniques that satisfy the above properties to an extent. Based on their functionality, these techniques can be classified as follows:

1. Signature-based detection (also known as Misuse detection):
   In this technique, a user's behaviour is compared with known attack patterns. If a match is found, an alert is raised. This type is capable of detecting only known attacks.

2. Specification-based detection:
   This technique inverts the signature-based approach [Wol06]; legitimate behaviour is specified and any deviation from legitimate behaviour raises an alert. The challenges in this approach include the complexities of many programs and the task of modeling such complex behaviour with precision. On the contrary, rough specification reduces the sensitivity of the detector.

3. Anomaly detection:
   This is the most promising technique of intrusion detection as it aims to detect novel attacks in addition to known attacks. In this type, observable behaviours of a system are used to build models for normal system operation [Lia05]. These behaviours may include audit logs, network sensors, system calls, etc. Various statistical techniques are used while building a model and also while classifying new instances. The drawback of this approach is the definition of *normal* behaviour. Expert domain knowledge may be required while making such profiles of normal behaviour.

Intrusion Detection systems can also be classified as network-based (which monitor network traffic) or host-based (which monitor operating system events). A more detailed description of these can be found in [MM01] and [Bac99].

As evident from the previous sections, anomaly detection is one of the most important and challenging tasks in the computer security domain. The remainder of this dissertation focuses on how machine learning and related ideas can be used to address this problem. In particular, we will see how decision making algorithms are capable of identifying anomalous behaviour by intelligent analysis of previous network behaviour.

Chapter 3

Machine Learning

This chapter describes the basics of machine learning. We first discuss about machine learning as a concept and thereafter describe the components and representation of a machine learning task in more formal terms.

3.1 Introduction

The concept of *learning* can be described in many ways including acquisition of new knowledge, enhancement of existing knowledge, representation of knowledge, organisation of knowledge and discovery of facts through experiments [MCM83]. When such learning is performed with the help of computer programs, it is referred to as *machine learning*. However, considering the difficulty in defining the measure of how much is learned, the authors in [WF05] have formulated an operational definition of machine learning:

> Things learn when they change their behaviour in a way that makes them perform better

This can be thought of learning in terms of performance (and not knowledge, which is rather abstract for computer programs), which is measurable in the case of computer programs. This performance may either be measured in terms of analysing the complexity of the algorithms or in terms of the functionality, the choice depending mainly on the area of application.

3.2 Basic Concepts

3.2.1 Learning

In computer science, every computer action can be modeled as a function with sets of inputs and outputs. A learning task may be considered as the estimation[1] of this function by observing the sets of inputs and outputs. The function estimating process usually consists of a *search in the hypothesis space* (i.e. the space of all such possible functions that might represent the input and output sets under consideration).

The authors in [Nil96] formally describe the function approximation process. Consider a set of input instances $X = (x_1, x_2, x_3...x_n)$. Let f be a function which is to be guessed by the learner. Let h be the learner's hypothesis about f. Also, we assume a priori that both f and h belong to a class of functions H. The function f maps the input instances in X as,

$$\mathrm{X} \overset{h \in H}{\rightarrow} \mathrm{h(X)}$$

A machine learning task may thus be defined as a search in this space H. This search results in *approximating* the relevant h, based on the training instances (i.e. the set X). The approximation is then checked against a set of test instances which are then used to indicate the correctness of h. The search requires algorithms which are efficient and which best-fit the training data [Mit97].

3.2.2 Knowledge Representation and Utilisation

Depending on the way in which the learned knowledge may be represented, machine learning may be divided into decision trees, neural networks, probability measures or other such representations. However, as identified in [DL03], a more fundamental way to divide machine learning is on the basis of the type of input and the way in which the learned knowledge is utilised. This division consists of:

- Learning for Classification and Regression: This is the most widely used method of learning involving classification and regression. Classification consists of assigning a new instance into one of the fixed classes from a finite set of classes. Regression involves the prediction of the new value on the basis of some continuous variable or attribute.

[1] We use the term *estimation* as the exact function may not be determinate.

- Learning for Acting and Planning: In this case, the learned knowledge is used for selecting an action for an agent. The action may be chosen in a purely *reactive* way, ignoring any past values. Alternatively, the output of classification or regression may be used by the agent to select an action based on the description of the current world state. These approaches are useful for problem solving, planning and scheduling.
- Learning for Interpretation and Understanding: This type focuses on the the interpretation and understanding of situations or events rather than just the accurate prediction of new instances. Many separate knowledge elements are used to derive this understanding, which is known as *abduction.*

3.2.3 Inputs and Outputs

The inputs and outputs to a machine learning task may be of different kinds. Generally, they are in the form of numeric (both discrete and real-valued) or nominal attributes. Numeric attributes may have continuous numeric values whereas nominal values may have values from a pre-defined set. For instance, an attribute like *temperature* if used as a numeric attribute, may have values like $25^oC, 28^oC$, etc. On the other hand, if it is used as a nominal attribute, it may take values from a fixed set (like high, medium, low). In many cases, the output may also be a boolean value.

3.3 Production of Knowledge

The way in which knowledge is learned is another important issue for machine learning. The learning element may be trained in different ways [DL03], supplementary to the classes identified in section 3.2.2. For classification and regression, knowledge may be learned in a *supervised*, *unsupervised* or *semi-supervised* manner. In the case of supervised learning, the learner is provided with training examples with the associated classes or values for the attribute to be predicted. *Decision-tree and rule induction methods, neural network methods, nearest neighbour approaches and probabilistic methods* are examples belonging to the type of supervised learning. These methods differ in the way they represent the learned knowledge (like rules, decision trees, probabilities, etc.) and also in the algorithms that are used for learning.

Unsupervised learning is concerned with the provision of training examples without any class association or any value for an attribute used for prediction. *Clustering* and *Density estimation* are examples of unsupervised learning approaches. In the case of clustering, the goal of learning is to assign training

instances to classes of its own invention which can then be used to classify new instances. Density estimation is used to build a model that predicts the probability of occurrence for specific instances.

A third approach, which is essentially between the two described above, is that of semi-supervised learning. In this type of learning, the set of training examples is mixed, i.e. for some instances the associated classes are present, whereas for others, they are absent. The goal in this case is to model a classifier or regressor that accurately predicts and improves its behaviour by using the unlabeled instances.

One important thing to note is that the ultimate goal of any machine learning task should be to *generalise* the data and not *summarise* it. In the latter case, there will be no learning outcome and the learning exercise will tend to be futile as predictions on future data will not be possible.

3.4 Defining a Machine Learning task

In general, a machine learning task can be defined formally in terms of three elements, viz. the learning experience E, the tasks T and the performance element P. [Mit97] defines a learning task more precisely as,

> A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E.

This representation of a machine learning task clearly defines the requirements. It gives an idea of what the machine learning problem is, and what are its learning goals. It also states how these goals can be measured so that the effectiveness of the task can be decided.

3.5 Life Cycle of a Machine Learning task

The life cycle of a machine learning task generally follows the process as depicted in Fig: 3.1.

1. Choosing a learning algorithm.
2. Training the algorithm using a set of instances (referred to as the *training set*).

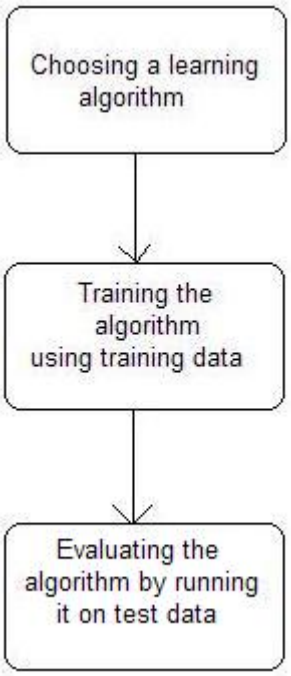


Figure 3.1: Machine Learning Flow

3. Evaluating the performance by running the algorithm on another set of instances (referred to as the *test set*).

Depending on the nature of the knowledge to be learned, different types of algorithms may be chosen at different times. Also, the type of inputs and outputs are also instrumental in choosing an algorithm. These include rule inferring algorithms, algorithms based on statistical models, divide and conquer approaches, covering algorithms, algorithms for mining association rules, algorithms based on linear models, algorithms on instance-based learning and clustering algorithms. Table 3.1 shows the typical algorithm choices based on the type of data available for learning [WF05].

Table 3.1: Selection of algorithms (adapted from [WF05])

Sr. No.	*Dataset Characteristics*	*Algorithm Choice*
1	A single attribute can generalise the data	Simple Rule inferring algorithms (e.g. The 1- Rule Algorithm [Hol93])
2	Allow all attributes to contribute equally and independently	Statistical Modeling (e.g. Naive Bayes [JL95])
3	Datasets having a simple logical structure which can be captured by constructing decision trees using a few attributes	Decision Trees (e.g. ID3 [Ges97], C4.5 [Qui93])
4	Independent rules might be able to assign the instances to different classes	Covering Algorithms (e.g. RIPPER [Coh95], PRISM [Cen87])
5	Dependence among subsets of attributes	Association Rules (e.g. Apriori [AS94])
6	Presence of numeric attributes with underlying linear dependencies requiring a weighted sum of attribute values with *appropriately* chosen weights	Linear Models (e.g. Perceptron Algorithm [FS99], Winnow [Lit88])
7	Presence of numeric attributes, in which classifications appropriate to particular regions of instance space might be governed by the *distances* between themselves	Euclidean Distance Algorithms (e.g. k-Nearest Neighbours [EPY97])
8	Datasets in which no class is to be predicted, but instances are to be divided into natural groups	Clustering (e.g. k-means [Mat00])

A detailed description of all these algorithms and their working logic is out of the scope of this thesis. Similarly, it is also non-trivial to compare these algorithms with regards to their relative advantages and disadvantages. For example, rule-based algorithms may generate simple rules, but they might not generalise well and also result in *overfitting*[1]. On the other hand, instance-based learners may involve a high cost in classifying new instances and clustering may require

[1] A machine learning method is said to overfit if it performs well on the training data but does not perform well on the test data.

a large dataset (as compared to other methods) to perform well [Lia05]. The interested reader may refer to [WF05] for a more comprehensive explanation of all these algorithms.

Once the algorithm is selected, the next step is to train the algorithm by providing it with a set of *training* instances. The training instances are used to build a model that represents the target concept to be learned (i.e. the hypothesis). As mentioned earlier, the main goal of this hypothesis formulation is to generalise the data to the maximum possible extent. This model is then evaluated using the set of *test* instances. It is extremely important to take care that the data instances used in the training process *must not* be used during the testing phase as it may lead to an overly optimistic model.

In the case where a large amount of data is available, the general approach is to construct two independent sets, one for training and the other for testing. On the other hand, if a limited amount of data is available, it becomes difficult to create separate sets for training and testing. In such cases, some data might be held over for testing, and the remaining used for training[1]. However, the data in this case might be distributed in an uneven way in the training and test sets and might not represent the output classes in the correct proportions. A method called *stratification* can be used to circumvent this problem. In stratification, the training and test sets are created in such a manner that the class distribution from the whole dataset is preserved in the training and test sets [Hal99]. This may be used in conjunction with another approach called *cross-validation* for an even better learning performance [Koh95]. For a *k-fold* cross-validation, the data is divided into k folds. The learning process is repeated k times, and every time a single fold is used for testing and the remaining *k-1* folds are used for training. The results of a learner are then averaged accordingly depending on the value of k.

3.6 Benefits of Machine Learning

The field of machine learning has been found to be extremely useful in the following areas relating to software engineering [Mit97]:

1. Data Mining problems where large databases may contain valuable implicit regularities that can be discovered automatically.

2. Difficult to understand domains where humans might not have the knowledge to develop effective algorithms.

[1] This is called the *holdout* procedure [WF05].

3. Domains in which the program is required to adapt to dynamic conditions.

In the case of traditional intrusion detection systems, the alerts generated are analysed by a human analyst who evaluates them and takes a suitable action. However, this is an extremely onerous task as the number of alerts generated may be quite large and the environment may change continuously [Pie04]. This makes machine learning well suited for intrusion detection.

The next chapter describes the application of machine learning to security, more precisely to intrusion detection, where two machine learning algorithms are analysed.

Chapter 4

Machine Learning applied to Computer Security

"In a world in which the total of human knowledge is doubling about every ten years, our security can rest only on our ability to learn"

-Nathaniel Branden

In this chapter, we focus on the application of machine learning to the field of detecting intrusions. We first define a machine learning task for detecting intrusions in formal terms as described in section 3.4. We then analyse two different machine learning algorithms, the VFI algorithm [DG97] and the NBTree algorithm [Koh96] with respect to their functioning, accuracy and efficiency in detecting novel intrusions. Being based on different strategies, these algorithms are expected to have different outcomes which will be compared.

4.1 Defining Intrusion Detection as a Machine Learning Task

A machine learning task can be formally defined as shown in section 3.4. In this section, we use this notation to formulate intrusion detection as a machine learning task[1]. The parameters used to define a machine learning task are the *experience* E, the *task* T and *performance measure* P.

For intrusion detection, the task T is the ability to detect intrusions as accurately as possible. More precisely, for the scope of this thesis, *good* and *bad*

[1]This task formulation holds within the scope of this dissertation. In other scenarios, it may be defined alternatively depending on the learning requirements.

connections should be identified correctly. The experience E is provided as a dataset. It consists of data that defines normal as well as abnormal behaviour. The data set used in this dissertation is described in more detail in section 4.3. Each of the instances in the training set provides some sort of an experience to the learner. The performance P is measured in terms of classification accuracy and other parameters like precision, recall, F-measure and kappa statistic which are described in section 4.5.2. Thus, for intrusion detection, we have,

1. Task: To detect intrusions in an accurate manner.
2. Experience: A dataset with instances representing normal as well as attack data.
3. Performance Measure: Accuracy in terms of correct classification of intrusion events and normal events and other statistical metrics including precision, recall, F-measure and kappa statistic.

4.2 Related Work

A comparison of eager, lazy and hybrid learning algorithms for improving intrusion detection has been done in [Tes07]. A dataset generated from the honeypots of Tilburg University was used for this study. RIPPER [Coh95] and IB1 [AKA91] were the respective eager and lazy learning methods used to learn intrusions from the data set. Attribute selection was done using InfoGain [Mit97] and CFS [Hal99] mechanisms. This attribute selection was done only during the IB1 selection as RIPPER has an inherent feature selection capability. A hybrid learning model was also used by combining the above two approaches. It was found that RIPPER was the most effective mechanism for classifying new intrusions on the concerned data set.

A new algorithm called LERAD was devised for learning useful system call attributes [TC05]. This is a conditional rule learning algorithm which generates a small number of rules. The rules can be effectively used to detect more attacks with a reasonable space and time overhead. In this research, it was shown that analysis of system call attributes is extremely useful in detecting attacks and it performs better than systems which analyse just the sequence of system calls. This methodology mainly targeted at making the IDS deterrent to mimicry attacks [WS02].

In [IYWL06], a combination of signature-based and machine learning-based intrusion detection systems is shown to be quite useful. The authors showed

that when a signature-based IDS is used as the main system and the machine learning-based IDS is used as a supporting system, the supporting system can filter out the false alarms and validate the decisions of the main system. Snort was used as the signature-based IDS and extended IBL (Instance-based Learner) [AKA91] was used as the algorithm in the machine learning-based IDS.

Another important research work was done in [Lia05], where the task of detecting intrusions was associated with a text mining task. The frequency of system calls was an important element in this thesis. The k-Nearest neighbour learning methodology [Mit97] was used to categorise intrusions based on the frequency of system calls. A cost-based model[1] for determining the interdependence between the IDS and the attacker was also presented, which was shown to be quite effective.

In [SS03], it was shown that certain attack categories are better detected by certain algorithms. A multi-classifier machine learning model using these individual algorithms was built and to detect attacks in the KDD 1999 Cup Intrusion Detection dataset. Nine different algorithms representing a variety of fields were selected for this analysis. Probability of detection and false alarm rate were used as the performance measures. Empirical results indicated that a noticeable performance improvement was achieved for certain probing, DoS and user-to-root attacks.

4.3 Data Set Description

The UCI machine learning repository [AN07] is one of the most comprehensive archives for the machine learning community. The data set used for evaluation in this dissertation is a subset of the KDD Cup '99 data set[2] for intrusion detection obtained from the UCI machine learning repository. This data set was used for *The Third International Knowledge Discovery and Data Mining Tools Competition* and also in *KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining.* The goal of the competition was to build a network intrusion detector, a predictive model capable of distinguishing between *intrusions (or attacks)* and *normal* connections. It contains a standard set of

[1] A cost-based model is one in which every decision taken by the learner is associated with a cost. This cost may be specific to the domain in which the learning has to take place. An example is described in [SJS00].

[2] The KDD Cup '99 data set is a version of a data set used at the DARPA Intrusion Detection Evaluation program. More details of the original data set can be found at *http* : *//www.ll.mit.edu/IST/ideval/data/data_index.html*

data which includes a wide variety of intrusions simulated in a military network environment [OC99c].

4.3.1 Characteristics of the Data Set

The data set consists of TCP dump data for a simulated Air Force LAN. In addition to normal LAN simulation, attacks were also simulated and the corresponding TCP data was captured. The attacks were launched on three UNIX machines, Windows NT hosts and a router along with background traffic. Every record in the data set represents a TCP connection i.e. a sequence of TCP packets starting and ending at some well defined times, between which data flows from a source IP address to a destination IP address under a well defined protocol. Each connection was labeled as normal or as a specific attack type [OC99b]. The attacks fall into one of the following categories:

- DOS attacks (Denial of Service attacks)
- R2L attacks (unauthorised access from a remote machine)
- U2R attacks (unauthorised access to super user privileges)
- Probing attacks

4.3.2 Features

In addition to the basic features (i.e. attributes) of the TCP dump data, the authors in [SJS00] have defined higher-level features[1] that help in identifying normal connections and attacks. The *'same host'* features examine connections in the past two seconds that have the same destination host as the current connection and calculate statistics including protocol behaviour, service, etc. The *same service* features are identical to the *same host* ones except that they have the same service as the current connection. These two features are referred to as *time-based* features. Another set of features called the *host-based* features were constructed using a window of 100 connections to the same host and the connection records were sorted by destination host. The last set of features called *content* features were constructed by using domain knowledge so as to look for suspicious behaviour (like number of failed logins) [SJS00]. Some of these features are nominal and some are continuous. A complete list of the set of these features for the connection records is explained in more detail in tables 4.1, 4.2 and 4.3. In addition to this, table 4.4 shows the exact attributes used in this dissertation

[1]The feature description is adapted from [OC99b].

along with their types. A sample entry in the dataset is also shown in table 4.5. The last attribute in thsi entry is the *Class* attribute which indicates whether the instance is normal or abnormal. It is specified as *normal* for normal instances and one of *buffer_overflow, loadmodule, perl, neptune, smurf, guess_passwd, pod, teardrop, portsweep, ipsweep, land, ftp_write, back, imap, satan, phf, nmap or multihop* for abnormal instances [OC99a].

Table 4.1: Basic Features [OC99b]

Feature Name	*Description*	*Type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Table 4.2: Content Features [OC99b]

Feature Name	*Description*	*Type*
hot	number of 'hot' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of 'compromised' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if 'su root' command attempted; 0 otherwise	discrete
num_root	number of 'root' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous

is_hot_login	1 if the login belongs to the 'hot' list; 0 otherwise	discrete
is_guest_login	1 if the login is a "guest"login; 0 otherwise	discrete

Table 4.3: Traffic Features [OC99b]

Feature Name	*Description*	*Type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 4.4: Dataset Attributes [OC99a]

Sr. No.	*Feature Name*	*Type*
1	duration	continuous
2	protocol_type	nominal
3	service	nominal
4	flag	nominal
5	src_bytes	continuous
6	dst_bytes	continuous
7	land	nominal
8	wrong_fragment	continuous
9	urgent	continuous
10	hot	continuous

11	num_failed_logins	continuous
12	logged_in	nominal
13	num_compromised	continuous
14	root_shell	continuous
15	su_attempted	continuous
16	num_root	continuous
17	num_file_creations	continuous
18	num_shells	continuous
19	num_access_files	continuous
20	num_outbound_cmds	continuous
21	is_host_login	nominal
22	is_guest_login	nominal
23	count	continuous
24	srv_count	continuous
25	serror_rate	continuous
26	srv_serror_rate	continuous
27	rerror_rate	continuous
28	srv_rerror_rate	continuous
29	same_srv_rate	continuous
30	diff_srv_rate	continuous
31	srv_diff_host_rate	continuous
32	dst_host_count	continuous
33	dst_host_srv_count	continuous
34	dst_host_same_srv_rate	continuous
35	dst_host_diff_srv_rate	continuous
36	dst_host_same_src_port_rate	continuous
37	dst_host_srv_diff_host_rate	continuous
38	dst_host_serror_rate	continuous
39	dst_host_srv_serror_rate	continuous
40	dst_host_rerror_rate	continuous
41	num_outbound_cmds	continuous
42	dst_host_srv_rerror_rate	continuous
43	Class	nominal

Table 4.5: A sample record of the dataset

Sr. No.	*Feature Name*	*Value*
1	duration	0

2	protocol_type	tcp
3	service	http
4	flag	SF
5	src_bytes	181
6	dst_bytes	5450
7	land	0
8	wrong_fragment	0
9	urgent	0
10	hot	0
11	num_failed_logins	0
12	logged_in	1
13	num_compromised	0
14	root_shell	0
15	su_attempted	0
16	num_root	0
17	num_file_creations	0
18	num_shells	0
19	num_access_files	0
20	num_outbound_cmds	0
21	is_host_login	0
22	is_guest_login	0
23	count	8
24	srv_count	8
25	serror_rate	8
26	srv_serror_rate	0
27	rerror_rate	0
28	srv_rerror_rate	0
29	same_srv_rate	0
30	diff_srv_rate	1
31	srv_diff_host_rate	0
32	dst_host_count	0
33	dst_host_srv_count	9
34	dst_host_same_srv_rate	9
35	dst_host_diff_srv_rate	1
36	dst_host_same_src_port_rate	0
37	dst_host_srv_diff_host_rate	0.11
38	dst_host_serror_rate	0
39	dst_host_srv_serror_rate	0

40	dst_host_rerror_rate	0
41	num_outbound_cmds	0
42	dst_host_srv_rerror_rate	0
43	Class	Normal

4.4 Algorithms

This section describes the algorithms used for learning intrusions. The first algorithm is the NBTree [Koh96], which is a hybrid of decision-tree and Naive Bayes classifiers [LIT92]. This is a significant improvement over the performance of a traditional Naive Bayes classifier in terms of its accuracy. The second algorithm is the VFI [DG97], which performs classification on the basis of real-valued vote distribution among classes.

4.4.1 NBTree

The NBTree algorithm is a hybrid between decision-tree classifiers and Naive Bayes classifiers. In order to understand the functioning of NBTree, it is first important to know how classification is done on the basis of decision-trees and Naive Bayes separately. The following sub-sections briefly explain the decision-tree and Naive Bayes approaches. Finally the NBTree approach is described.

4.4.1.1 Decision Tree Classification

In this type of classification, the target concept is represented in the form of a tree. The tree is built by using the principle of recursive partitioning. An attribute is selected as a partitioning attribute (also referred to as node) based on some criteria (like information gain) [Mit97]. This process is then repeated for every child node until all attributes are exhausted and a decision-tree is constructed. This decision tree may be further pruned so as to reduce the tree size [Qui93] and also to avoid overfitting [Mit97].

4.4.1.2 Naive Bayes Classification

Naive Bayes classification [LIT92] is based on Bayesian probability learning. It assumes conditional independence of attributes given the target value of the instance. For a learning task based on Naive Bayes classification, each instance in the dataset is represented as a conjunction of attribute values where the target function $f(x)$ can take values from a finite set V. A new instance to be classified

is presented in the form of a tuple $(a_1, a_2, a_3...a_n)$ and the learner is supposed to classify this new instance. The output v_{NB} of the Naive Bayes classifier is given by:

$$v_{NB} = \underset{v_j \in V}{argmax} \; P(v_j) \prod_i P(a_i|v_j)$$

The above probabilities are based on the frequencies of attribute values in the training data. In this case, there is no explicit search in the hypothesis space, just frequency counting. A more detailed explanation can be found in [Mit97].

4.4.1.3 The NBTree Approach

The Naive Bayes method performs quite well for discrete-valued attributes. As the data size increases, the performance improves further. However, for continuous-valued attributes, it does not take into account attribute interactions [Koh96]. Decision-trees on the other hand do not give a good performance when the data size is extremely large. These two weaknesses are overcome by the NBTree algorithm.

The NBTree algorithm represents the learned knowledge in the form of a tree which is constructed recursively. However, the leaf nodes are Naive Bayes categorizers rather than nodes predicting a single class [Koh96]. For continuous attributes, a threshold is chosen so as to limit the entropy measure. The utility of a node is evaluated by discretizing the data and computing the 5-fold cross-validation accuracy estimation using Naive Bayes at the node. The utility of the split is the weighted sum of utility of the nodes and this depends on the number of instances that go through that node. The NBTree algorithm tries to approximate whether the generalisation accuracy of Naive Bayes at each leaf is higher than a single Naive Bayes classifier at the current node. A split is said to be significant if the relative reduction in error is greater that 5% and there are atleast 30 instances in the node [Koh96]. The NBTree pseudo-code is shown as Algorithm 1.

The authors in [Koh96] showed that NBTree has considerable performance improvement above its individual constituents and is suited to many real world datasets. For n attributes, m attributes and l classes, the complexity of NBTree for the attribute selection phase (for discretized attributes) is $O(m \cdot n^2 \cdot l)$. When the number of attributes is less than $O(log\, m)$ (which is the usual case), and the number of labels is small, then the time spent on attribute selection using cross-validation is less than the time spent in sorting the instances by each attribute. NBTree thus scales up well to large databases.

Algorithm 1 The NBTree Algorithm [Koh96]

Input: a set of T labeled instances

Output: a decision-tree with Naive Bayes categorizers at leaves

1. For each attribute X_i, evaluate the utility $u(X_i)$, of a split on attribute X_i. For continuous attributes, a threshold is also found out at this stage.
2. Let $j = argmax_i(u_i)$, i.e. the attribute with the highest utility.
3. If u_j is not significantly better than the utility of the current node, create a Naive Bayes classifier for the current node and return.
4. Partition T according to the test on X_j. If X_j is continuous, a threshold split is used; if X_j is discrete, a multi-way split is made for all possible values.
5. For each child, call the algorithm recursively on the portion of T that matches the test leading to the child.

Another advantage of NBTree is its simplicity of representing the learned knowledge. It segments the data using a univariate decision tree, thus making the segmentation easy to understand. Every leaf being a Naive Bayes classifier, can also be easily understood [Koh96].

4.4.2 VFI

The VFI[1] algorithm is a classification algorithm based on the voting frequency intervals (hence the name VFI). In VFI, each training instance is represented as a vector of features along with a label that represents the class of the instance. Feature intervals are then constructed for each feature. An interval represents a set of values for a given feature where the same subset of class values are observed. Thus, two adjacent intervals represent different classes.

In the training phase of VFI, the feature intervals are calculated by calculating the lowest and highest feature value for every linear feature for every class. For nominal features, the observed feature values are taken into consideration. For each linear feature with k classes, $2k$ values are found. These values are then sorted and each pair of consecutive points forms a feature interval. Point intervals are formed for nominal values. Each of these intervals is represented as a vector $(lower, count_1, count_2....count_k)$, where $lower$ is the lowest feature value and $count_i$ is the number of training instances of class i that fall into that interval.

In the classification phase, the interval i of the new instance e is found out. A feature vote is then calculated for every class as follows:

[1] This explanation is adapted from [DG97]

$$feature\_vote[f, c] = \frac{Number\ of\ instances\ of\ class\ c\ which\ fall\ into\ interval\ i\ of\ feature\ f}{total\ number\ of\ classes}$$

These votes are then normalised and the the class with the highest feature vote is returned as the predicted class for the new instance. The following Algorithm 2 gives a simplified picture of the actual VFI algorithm.

Algorithm 2 The VFI Algorithm (adapted from [DG97])

Input: a set of T labeled instances

Output: The predicted class for a given new instance

1. Training Phase

For each feature, find the lowest and highest feature value for every linear class and the observed value for every nominal class. Calculate the feature intervals based on these values.

For each class, calculate the number of training instances that fall into every interval.

2. Classification Phase

For a new instance e to be classified, find the interval into which the value of e falls. Calculate the vote given by each feature, normalize the votes and return the class with the highest vote.

4.5 Experimental Analysis

The experiments done in light of the information explained in the previous sections of this thesis consist of the evaluation of the performance of NBTree and VFI algorithms for the task of classifying novel intrusions. The dataset described in section 4.3 was used in the experiments. Weka [WF05], a machine learning toolkit was used for the implementation of the algorithms described in sections 4.4.1 and 4.4.2. Weka provides a common interface for all the algorithm implementations that it contains. This interface can be used to specify the variables of an algorithm, to graphically visualise the dataset under consideration and to view the outcomes of algorithms on different datasets. This makes it convenient to compare various machine learning methods on a given dataset and to draw conclusions about the same. All resources used for this experiment including Weka and the data set are made available in Appendix C.

4.5.1 Environment

The experiments were carried out on a laptop with 1.60Ghz of dual core processing power and 2GB RAM. Due to the limitation in the available memory and processing power, it was not possible to use the full dataset described in section

4.3. Instead a reduced subset was used and 10-fold cross-validation (explained in section 3.5) was used to overcome this limitation.

4.5.2 Evaluation Metrics

In order to analyse and compare the performance of the above mentioned algorithms, metrics like the classification accuracy, precision, recall, F-Measure and kappa statistic were used. These metrics are derived from a basic data structure called as the *confusion matrix*. A sample confusion matrix for a two-class problem can be represented as:

Table 4.6: Confusion Matrix for a two-class problem(adapted from [WF05])

	Predicted Class Positive	*Predicted Class Negative*
Actual Class Positive	a	b
Actual Class Negative	c	d

In this confusion matrix, the value a is called a *true positive* and the value d is called a *true negative*. The value b is referred to as a *false negative* and c is known as *false positive* [Wol06].

In the context of intrusion detection, a true positive is an instance which is normal and is also classified as normal by the intrusion detector. A true negative is an instance which is an attack and is classified as an attack.

4.5.2.1 Classification Accuracy

This is the most basic measure of the performance of a learning method. This measure determines the percentage of correctly classified instances. From the confusion matrix, we can say that:

$$Accuracy = \frac{a+d}{a+b+c+d}$$

This metric gives the number of instances from the dataset which are classified correctly i.e. the ratio of true positives and true negatives to the total number of instances.

4.5.2.2 Precision, Recall and F-Measure

Precision and recall [WMB99] are terms used widely in the information retrieval domain. They are usually defined as:

Precision = ratio of number of documents retrieved that are relevant to the total number of documents that are retrieved
Recall = ratio of number of documents retrieved that are relevant to the total number of documents that are relevant

Precision gives the percentage of slots in the hypothesis that are correct, whereas recall gives the percentage of reference slots for which the hypothesis is correct. Precision takes into account the substitution and insertion errors whereas recall takes into account the substitution and deletion errors [MKSW99].

Referring from the confusion matrix, we can define precision and recall for our purposes as [Tes07]:

$$Precision = \frac{a}{a+c}$$
$$Recall = \frac{a}{a+b}$$

The precision of a intrusion detection learner would thus indicate the proportion of total number of correctly classified positive instances to the total number of predicted positive instances and recall would indicate the proportion of correctly classified positive instances to the total number of actual positive instances. Thus, a high precision and high recall are desirable for an IDS.

The authors in [KM97] identify that accuracy alone cannot be considered as sole reliable measure for classification. This is because in a case where there are 10 instances, out of which 9 are negative and 1 is positive, if the classifier classifies all of them as negative, the accuracy would be 90%. However, it would result in ignoring all the positive instances. The F-measure is therefore defined as the weighted harmonic mean of precision and recall [MKSW99] to address this problem which may be present in any classification scenario.

$$F\text{-}measure^{1} = \frac{2*Precision*Recall}{Precision+Recall}$$

4.5.2.3 Kappa Statistic

The Kappa statistic is used to measure the agreement between predicted and observed categorizations of a dataset, while correcting for agreements that occur by chance. It takes into account the expected figure and deducts it from the predictor's success and expresses the result as a proportion of the total for a perfect predictor [WF05]. So, for instance, if we had another confusion matrix for the expected predictions of classes as follows:

[1] This is the most popular formula for the F-measure. A more general formula can be found in [MKSW99].

Table 4.7: Confusion Matrix for a two-class problem (Expected predictions)

	Predicted Class Positive	*Predicted Class Negative*
Actual Class Positive	e	f
Actual Class Negative	g	h

the Kappa statistic is calculated as,

$$Kappa = \frac{(a+d)-(e+h)}{(a+b+c+d)-(e+h)} \text{ x } 100$$

In addition to the above statistical metrics, the time taken to build the model was also considered as a performance indicator.

4.5.3 Attribute Selection

Attribute Selection (also known as Feature Reduction or Feature Subset Selection) is an important task during any machine learning exercise (especially classification). Usually, the available data for machine learning analysis is multi-dimensional and the number of dimensions (i.e. features or attributes) is often high. It is not necessary that all the features of a particular dataset are important for a machine learning algorithm. Many features may be redundant and may not contribute at all to the classification task. For a dataset with k attributes, the size of the hypothesis space is k^2. For a small k, an exhaustive search to find out the best suited hypothesis is possible. However, this task becomes non trivial as the value of k increases [HS96].

Feature subset selection is the process of identifying and removing much of the redundant and irrelevant information possible. This results in the reduction of dimensionality of the data and thereby makes the learning algorithms run in a faster and more efficient manner. It also reduces the size of hypothesis space and in some cases, it also reduces the storage requirement [Hal99].

The experiments conducted in this thesis use the *Information Gain* attribute selection method. This method is explained in the following sub-section.

4.5.3.1 Information Gain

Information Gain measure is used to determine how instrumental is a particular attribute in correctly classifying the training data. Information gain is based on the concept of *entropy* which is widely used in the information theory domain. Given a collection of instances S, containing positive and negative examples of some target concept. the entropy of S relative to this boolean classification is given by [Mit97]:

$$Entropy(S) = -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

where $p_{\oplus}$ is the proportion of positive examples in S and $p_{\ominus}$ is the proportion of negative examples in S.

For a target concept with c different possible values for the classes, the entropy can be defined as [Mit97]:

$$Entropy(S) = \sum_{i=1}^{c} - p_i log_2 p_i$$

where p_i is the proportion of S belonging to class i.

Based on the above definition of entropy, information gain G of an attribute A is defined as [Mit97]:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where Values(A) is the set of all possible values for attribute A and S_v is the subset of S for which A has value v. Information gain is thus the reduction in entropy caused by partitioning the examples according to an attribute.

In this thesis, Weka's implementation of the Information gain attribute selector (called InfoGainAttributeEval) was used to determine the effectiveness of attributes and the attributes were ranked in decreasing order of information gain values. The first 7 attributes and the *Class* attribute were then used in the experiment for the learning task. Table 4.8 lists these selected attributes which is a subset of those shown in Table 4.5.

Table 4.8: Attributes selected using Information Gain

Sr. No.	*Feature Name*	*Type*
1	service	nominal
2	src_bytes	continuous
3	dst_bytes	continuous
4	logged_in	nominal
5	count	continuous
6	srv_count	continuous
7	dst_host_diff_srv_rate	continuous
8	Class	nominal

4.5.4 Summary of Experiments

Based on the above algorithms, attribute selection methods and the type of cross-validation, the following table shows a summary of the experiments conducted.

Table 4.9: Summary of Experiments

Algorithm	*Feature Reduction Method*	*Cross-Validation*
NBTree	-	10-fold
VFI	-	10-fold
NBTree	Information Gain	10-fold
VFI	Information Gain	10-fold

Chapter 5

Results

The results[1] of the experiments described in section 4.5 are discussed in this chapter. A comparison between NBTree and VFI methods is also made based on the values of the metrics defined in section 4.5.2. 10-fold cross-validation was used for all the experiments. These results are then interpreted and conclusions are drawn based on this analysis as to which of the methods is best suited to solve the intrusion detection problem at hand.

5.1 Results of NBTree

NBTree was evaluated by taking into account all features of the dataset. The results of this evaluation are summarised in the table below. The detailed result set is provided in Appendix A.

Table 5.1: Results of NBTree with all attributes

Metric	*Value*
Time taken to build the model	1115.05s
Accuracy	99.94 %
Average Precision	90.33 %
Average Recall	92.72 %
Average F-Measure	91.14 %
Kappa Statistic	99.99 %

NBTree was further evaluated on the dataset by taking into account feature reduction using the Information Gain measure. The results of this test are sum-

[1]The decimal quantities in this section are rounded to 2 decimal places for consistency.

marised in the following table.

Table 5.2: Results of NBTree with selected attributes using Information Gain measure

Metric	*Value*
Time taken to build the model	38.97s
Accuracy	99.89 %
Average Precision	94.54 %
Average Recall	90.84 %
Average F-Measure	92.28 %
Kappa Statistic	99.82 %

The NBTree clearly has a very high accuracy rate of classifying almost 99% of the instances correctly in both cases (i.e. when all attributes were considered and when only a selected attributes were considered). Also, it has high precision and recall values, which are important in the intrusion detection task. The time taken by NBTree for constructing the model is however greater than that of VFI in all cases.

The feature selection mechanism does not affect the classification accuracy, precision and recall values of NBTree to a great extent. In fact, the results show that the classification accuracy is slightly higher with all attributes taken into consideration. However, this is accompanied with a steep rise in the time taken to build the model. With selected attributes, the time taken is 38.97s whereas it is 1115.05s when all attributes are considered. This shows that NBTree is extremely time-sensitive with regards to the number of attributes used. The F-Measure also has a considerably high value (above 90% in both cases) so that the problem described in section 4.5.2.2 is addressed to a good extent.

5.2 Results of VFI

Similar to the NBTree tests, VFI was also evaluated twice; once by taking all attributes into consideration and then by using a reduced attribute subset obtained by the Information Gain measure. The results of these experiments are listed in the following tables. A detailed result set is provided in Appendix B.

Table 5.3: Results of VFI with all attributes

Metric	*Value*
Time taken to build the model	0.92s
Accuracy	86.58 %
Average Precision	41.27 %
Average Recall	80.54 %
Average F-Measure	44.05 %
Kappa Statistic	79.50 %

Table 5.4: Results of VFI with selected attributes using Information Gain measure

Metric	*Value*
Time taken to build the model	0.2s
Accuracy	75.81 %
Average Precision	35.71 %
Average Recall	75.82 %
Average F-Measure	37.43 %
Kappa Statistic	66.21 %

The results indicate that VFI does not perform too well on the given dataset. It suffers from a low accuracy rate of just about 75% as compared to approximately 99% accuracy exhibited by NBTree. It also has very low precision values and low recall values. The F-Measure and Kappa statistic also have very low values. It is however faster than NBTree with regards to building the classification model.

It is observed that the feature selection mechanism reduces the classification accuracy of the VFI algorithm accompanied with a reasonable decrease in the time taken to build the model. The results indicate that performance is better in most aspects if all the dataset attributes are taken into account while classifying with VFI.

5.3 Interpretation of Results

For an IDS, the accuracy indicates how correct is the algorithm in identifying normal and adversary behaviour. Recall would indicate the proportion of cor-

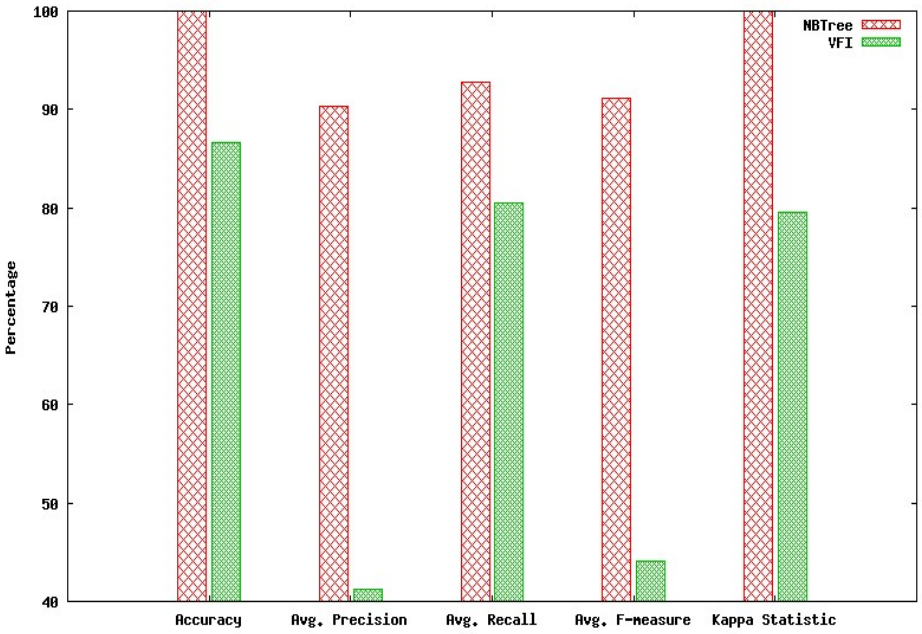


Figure 5.1: NBTree v/s VFI - All Attributes

rectly classified normal instances whereas precision would indicate the number of correctly classified normal instances from the total number of normal and attack instances. The Kappa statistic is a general statistical indicator and the F-Measure is related to the problem described in section 4.5.2.2. In addition to these, the time taken by the learning algorithm for model construction is also important as it may have to handle extremely large amounts of data.

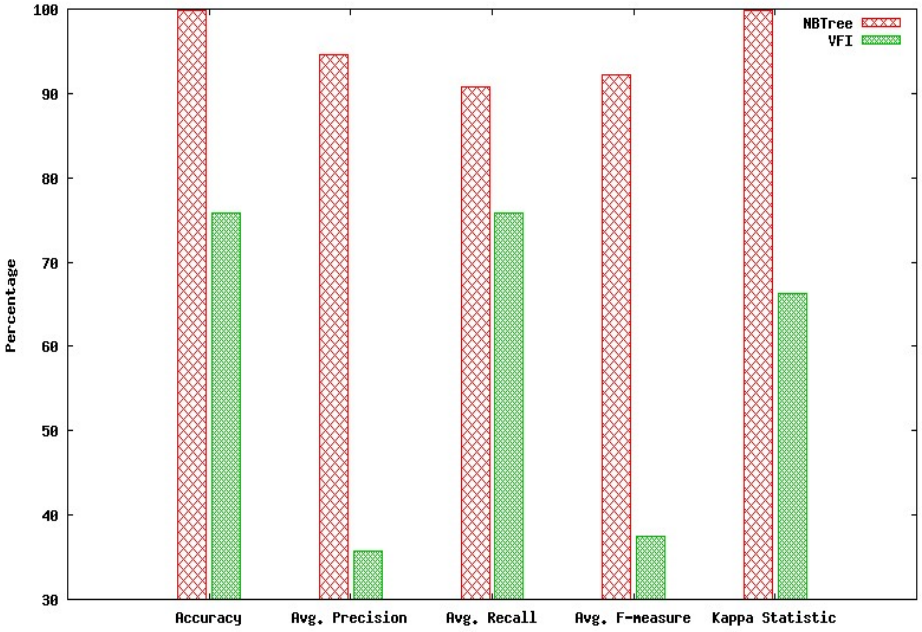


Figure 5.2: NBTree v/s VFI - Selected Attributes

The graphs in Figures 5.1 and 5.2 show a relative performance of NBTree and VFI for the intrusion detection task on the dataset under consideration. Figure 5.1 shows the comparison with all attributes under consideration and figure 5.2 depicts the comparison for attributes selected using the Information Gain measure.

As per the definitions in section 4.5.2.2, a good IDS should have a recall that is as high as possible. A high precision is also desired. From our results, we see that the classification accuracy of NBTree is better than that of VFI in both cases. There are tremendous differences in the precision and recall values of NBTree and VFI where the NBTree exhibits a relatively higher precision and higher recall. Also, when all attributes are used, NBTree has a lower precision value than the case when selected attributes are used. In both these cases, the recall is more or less the same. Also, the F- Measure value is high for NBTree in comparison to VFI. NBTree is seen to have a better performance as compared to the VFI in both the cases and it can thus be said that it is more suited to the intrusion detection task on the given dataset.

Chapter 6

Conclusions

In this thesis, we described the applications of machine learning to computer security, particularly to the task of detecting intrusions. Using Weka, we analysed two algorithms towards their suitability for detecting intrusions from a dataset containing audit data. We showed that machine learning can be effectively applied to detect novel intrusions and focused on anomaly detection. The two learning algorithms, NBTree and VFI were compared at the task of detecting intrusions. NBTree with an accuracy rate of approximately 99% and a recall of approximately 90% was found to perform much better at detecting intrusions than VFI for the dataset under consideration.

Based on the experiments done in the thesis and their corresponding results, we can state the following:

- Machine learning is an effective methodology which can be used in the field of Computer Security.
- The inherent nature of machine learning algorithms makes them more suited to the intrusion detection field of information security. However, it is not limited to intrusion detection. The authors in [MX06] have developed a tool using machine learning to infer access control policies where policy requests and responses are generated by using learning algorithms. These are effective with new policy specification languages like XACML [Mos05]. Similarly, a classifier-based approach to assigning users to roles and vice-versa is described in [SO04].
- It is possible to analyse huge quantities of audit data by using machine learning techniques, which is otherwise an extremely difficult task.

The following sections suggest some recommendations for future work, mention a real world application of machine learning to information security and also

discuss briefly on the security of machine learning itself.

6.1 Future Work

Machine Learning is an experimental science [Mit97]. A learning method which may be suited to a particular problem may not necessarily perform well at another problem. Also, a learning method may have many configurable parameters, which may result in a different performance.

Based on the above facts, the future work to this thesis can be summarised as follows:

1. In this thesis, two learning algorithms were tested and compared. The Weka machine learning toolkit offers a collection of many other learning schemes, which can be tested and evaluated.

2. The parameters of the machine learning schemes used in this thesis were default. It may be possible to further improve the performance of these schemes towards intrusion detection by optimizing these parameters.

3. Owing to the limited processing power, memory available for the experiments conducted and the scope of the thesis, a reduced subset of the actual dataset was used. These experiments can be repeated by taking the entire dataset which may further improve the performance of the learner.

4. The attribute selection mechanism (Section 4.5.3) used in this thesis was based on the Information Gain concept. Also, the top 7 attributes with maximum information gain (plus the class attribute) were used by the algorithms. It is possible to conduct experiments in which a different attribute selection mechanism is used and also, different number of attributes are selected to be given as inputs to the algorithms.

Thus, in order to realise the full potential of machine learning to the field of computer security, it is essential to experiment with various machine learning schemes towards addressing security-related problems and choose the one which is the most appropriate to the problem at hand.

6.2 Real-world Application

The Minnesota Intrusion Detection System (MINDS)[1] [EEL+] is an anomaly IDS implemented at University of Minnesota. It uses a set of machine learning techniques to automatically detect attacks against computer networks and systems. A density based outlier detection algorithm called *LOF* [BKNS00] is used in the anomaly detection module of MINDS. Experimental results involving live network traffic at the university have shown that it it extremely promising and successful at detecting several novel attacks which escaped identification from popular signature-based tools such as Snort[2].

6.3 Security of Machine Learning

While machine learning techniques are portraying a promising picture towards solving problems in computer security, it is imperative to consider how secure are the machine learning techniques themselves. The authors in [BNS+06] attempt to answer a few questions relating to the security of machine learning techniques. These questions quoted from [BNS+06] include:

- Can the adversary manipulate a learning system to permit a specific attack?
- Can an adversary degrade the performance of a learning system to the extent that system administrators are forced to disable the IDS?
- What defenses exist against adversaries manipulating (attacking) learning systems?
- What is the potential impact from a security standpoint of using machine learning on a system? Can an attacker exploit properties of the machine learning technique to disrupt the system?

Different attack scenarios with regards to subverting the learner were discussed. An adversary with deep understanding of the learning algorithm can be considered as a potential threat to the learning process. As an instance, an attack may cause the learner to build a model which deliberately misclassifies events. For addressing the above attacks, various defense mechanisms were also proposed.

[1] http://www.cs.umn.edu/research/MINDS/MINDS.htm

[2] http://www.snort.org/

Does machine learning for security introduce a vulnerability? Although the attacks and defenses in [BNS+06] were theoretical, current research in this field is increasingly aiming to ensure that the answer to this question is no.

Appendix A

NBTree Complete Results

Detailed results of the intrusion classification task using NBTree are available in a compressed format at the following URL:

`http://www.isg.rhul.ac.uk/~mrai182/Detailed%20NBTree%20Results.zip`

Appendix B

VFI Complete Results

Detailed results of the intrusion classification task using VFI are available in a compressed format at the following URL:

`http://www.isg.rhul.ac.uk/~mrai182/Detailed%20VFI%20Results.zip`

Appendix C

Experiment Resources

The resources used for this experiment are available in a compressed format at the following URL:

`http://www.isg.rhul.ac.uk/~mrai182/ExperimentResources.zip`

References

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, January 1991.

[AN07] A. Asuncion and D.J. Newman. UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[Bac99] Rebecca G. Bace. *Intrusion Detection*. Sams, December 1999.

[BCH+01] Eric Bloedorn, Alan D. Christiansen, Willian Hill, Clement Skorupka, Lisa M. Talbot, and Jonathan Tivel. Data mining for network intrusion detection: How to get started. `http://citeseer.ist.psu.edu/bloedorn01data.html`, Aug 2001.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, January 1984.

[BH95] Philippe Besnard and Steve Hanks, editors. *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, August 18-20, 1995, Montreal, Quebec, Canada*. Morgan Kaufmann, 1995.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, 2000.

[BNS+06] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06:*

Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pages 16–25, New York, NY, USA, 2006. ACM Press.

[Cen87] Jadzia Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[CER05] Insider threat study:computer system sabotage in critical infrastructure sectors. `http://www.cert.org/archive/pdf/insidercross051105.pdf`, 2005.

[CER07] CERT Vulnerability Statistics 1995 - 2006. `http://www.cert.org/stats/vulnerability_remediation.html`, 2007.

[Coh95] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[Cra06] Jason Crampton. Notes on Computer Security, 2006.

[CW87] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *IEEE Security and Privacy*, 00:184, 1987.

[DG97] Gulsen Demiroz and H. Altay Guvenir. Classification by voting feature intervals. In *European Conference on Machine Learning*, pages 85–92, 1997.

[DL03] Tom Dietterich and Pat Langley. Machine learning for cognitive networks:technology assessments and research challenges, Draft of May 11, 2003. `http://web.engr.oregonstate.edu/~tgd/kp/dl-report.pdf`, 2003.

[EEL+] Levent Ertz, Eric Eilertson, Aleksandar Lazarevic, Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. Minds - minnesota intrusion detection system. `http://www.cs.umn.edu/research/MINDS/papers/minds_chapter.pdf`.

[EPY97] Eppstein, Paterson, and Yao. On nearest neighbor graphs. *GEOMETRY: Discrete & Computational Geometry*, 17, 1997.

[FHSL96] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[FLSM00] Wei Fan, Wenke Lee, Salvatore J. Stolfo, and Matthew Miller. A multiple model cost-sensitive approach for intrusion detection. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 142–153. Springer, Berlin, 2000.

[FS99] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, December 1999.

[Ges97] Paul Gestwicki. Id3: History, implementation, and applications. `http://citeseer.ist.psu.edu/gestwicki97id.html`, 1997.

[Gol99] Dieter Gollmann. *Computer Security*. John Wiley & Sons, 1999.

[GSS99] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *ID'99: Proceedings of the 1st conference on Workshop on Intrusion Detection and Network Monitoring*, pages 6–6, Berkeley, CA, USA, 1999. USENIX Association.

[Hal99] Mark A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato, Department of Computer Science, 1999.

[Hol93] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90, April 1993.

[HS96] M. Hall and L. Smith. Practical feature subset selection for machine learning. In *Proceedings of the Australian Computer Science Conference*, 1996.

[IYWL06] Doo Heon Song Ill-Young Weon and Chang-Hoon Lee. Effective intrusion detection model through the combination of a signature-based intrusion detection system and a machine learning-based intrusion detection system. *Journal of Information Science and Engineering*, 22(6):1447–1464, 2006.

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[Ken99] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems. `http://www.kkendall.org/files/thesis/krkthesis.pdf`, 1999.

[KM97] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: one-sided selection. In *Proc. 14th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[Koh95] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1137–1145, 1995.

[Koh96] Ron Kohavi. Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207, 1996.

[KT03] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. `http://www.auto.tuwien.ac.at/~chris/research/doc/2003_03.ps`, 2003.

[Lan00] Terran D. Lane. *Machine Learning Techniques for the computer security domain of anomaly detection.* PhD thesis, Department of Electrical and Computer Engineering, Purdue University, August 2000.

[LB97a] T. Lane and C. Brodley. Detecting the abnormal: Machine learning in computer security. `citeseer.ist.psu.edu/lane97detecting.html`, 1997.

[LB97b] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 366–380, 1997.

[Lia05] Yihua Liao. *Machine Learning in Intrusion Detection.* PhD thesis, University of California (Davis), Department of Computer Science, 2005.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[LIT92] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *National Conference on Artificial Intelligence*, pages 223–228, 1992.

[LS00] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *Information and System Security*, 3(4):227–261, 2000.

[Mah03] M. Mahoney. *A Machine Learning Approach to Detecting Attacks by Identifying Anomalies in Network Traffic*. PhD thesis, Florida Institute of Technology, 2003.

[Mal06] Marcus A. Maloof, editor. *Machine Learning and Data Mining for Computer Security*. Springer, 2006.

[Mat00] Jiri Matousek. On approximate geometric k-clustering. *Discrete & Computational Geometry*, 24(1):61–84, 2000.

[MC02] M. Mahoney and P. Chan. Learning models of network traffic for detecting novel attacks. `http://www.cs.fit.edu/~mmahoney/paper5.pdf`, 2002.

[MCM83] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, 1983.

[MH03] Steve Moyle and John Heasman. Machine learning to detect intrusion strategies. *Knowledge-Based Intelligent Information and Engineering Systems*, 2773/2003:371–378, 2003.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MKSW99] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction. `http://www.nist.gov/speech/publications/darpa99/html/dir10/dir10.htm`, 1999.

[MM01] Ludovic M'e and C'edric Michel. Intrusion detection: A bibliography. Technical Report SSIR-2001-01, Sup'elec, Rennes, France, September 2001.

[Mos05] Tim Mose. Oasis, extensible access control markup language, (xacml) version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.

[MX06] Evan Martin and Tao Xie. Inferring access-control policy properties via machine learning. In *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 235–238, Washington, DC, USA, 2006. IEEE Computer Society.

[Nil96] Nils J.. Nilsson. Introduction to Machine Learning - an early draft of a proposed book. http://ai.stanford.edu/~nilsson/MLDraftBook/MLBOOK.pdf, 1996.

[NIS85] NIST. Trusted computer system evaluation criteria (orange book). http://csrc.nist.gov/publications/history/dod85.pdf, 1985.

[OC99a] University Of California. Intrusion detection dataset in machine readable form. http://kdd.ics.uci.edu/databases/kddcup99/kddcup.names, 1999.

[OC99b] University Of California. The UCI KDD Archive, University of California. http://kdd.ics.uci.edu/databases/kddcup99/task.html, 1999.

[OC99c] University Of California. The UCI KDD Archive, University of California. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[Pie04] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. *Recent Advances in Intrusion Detection*, 3224:102–124, 2004.

[PP03] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Pearson Education, Inc, 2003.

[PP07] Animesh Patcha and Jung-Min Park. Network anomaly detection with incomplete audit data. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(13):3935–3955, 2007.

[PT05] Tadeusz Pietraszeka and Axel Tannera. Data mining and machine learning—towards reducing false positives in intrusion detection. *Information Security Technical Report*, 10(3):169–183, 2005.

[Qui93] Ross R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[Ren04] Jason D. M. Rennie. Derivation of the f-measure. `http://people.csail.mit.edu/jrennie/writing/fmeasure.pdf`, Feb 2004.

[SJS00] Wenke Lee Salvatore J. Stolfo, Wei Fan. Cost-based modeling for fraud and intrusion detection results from the jam project. `http://www.cs.columbia.edu/~wfan/papers/costdisex.ps.gz`, 2000.

[SL06] Surendra K. Singhi and Huan Liu. Feature subset selection bias for classification learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 849–856, New York, NY, USA, 2006. ACM Press.

[SO04] Shengli Sheng and Sylvia L. Osborn. A classifier-based approach to user-role assignment for web applications. In *Secure Data Management*, pages 163–171, 2004.

[SS03] Maheshkumar Sabhnani and Gursel Serpen. Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context. In *Proceedings of International Conference on Machine Learning: Models, Technologies, and Applications*, pages 209–215, Las Vegas, Nevada, USA, 2003.

[Sta06] William Stallings. *Network Security Essentials: Applications and Standards (3rd Edition)*. Prentice Hall, 2006.

[TC05] G. Tandon and P. Chan. Learning useful system call attributes for anomaly detection. *Proc. 18th Intl. FLAIRS Conf.*, pages 405–410, 2005.

[Tes07] Sebastiaan Tesink. Improving intrusion detection systems through machine learning. `http://ilk.uvt.nl/downloads/pub/papers/thesis-tesink.pdf`, 2007.

[VMV05] Fredrik Valeur, Darren Mutz, and Giovanni Vigna. A learning-based approach to the detection of SQL attacks. In *DIMVA*, pages 123–140, 2005.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining - Practical Machine Learning Tools and Techniques, Second Edition*. Elsevier, 2005.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[Wol06] Stephen Wolthusen. Lecture 11 - Intrusion Detection and Prevention, notes in Network Security, 2006.

[WS02] D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems. `http://www.cs.berkeley.edu/~daw/papers/mimicry.pdf`, 2002.

[WZA06] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, 2006.